

LA-UR- 01-5065

Approved for public release;  
distribution is unlimited.

*Title:* Capturing Network Traffic with a MAGNeT

*Author(s):* J. R. Hay, W. Feng, and M. K. Gardner

*Submitted to:* The 5th Annual Linux Showcase & Conference (ALS 2001)

## Los Alamos

NATIONAL LABORATORY

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Form 836 (10/96)

LOS ALAMOS NATIONAL LABORATORY



3 9338 00789 3786

Current network traffic trace generation tools are able to monitor network traffic as it appears on the network, but are incapable of capturing traffic as it progresses through a running protocol stack. Thus, the current generation of tools can not record true application traffic demands, and are incapable of capturing run-time protocol state information to help fine-tune network performance. They also lend no insight into the modulating behavior of some protocols (e.g., TCP), which has been shown to have significant impact on network performance.

In this paper, we introduce the concept of a network application traffic trace generation tool. We discuss the need for such a tool, and the necessary modifications to the Linux kernel to enable such a tool. We also describe our own implementation, which we call the Monitor for Application-Generated Network Traffic (MAGNeT), and propose several ways in which MAGNeT can help increase network performance.

# Capturing Network Traffic with a MAGNeT

Jeffrey R. Hay, Wu-chun Feng, Mark K. Gardner  
{feng, jrhay, mkg}@lanl.gov  
*Advanced Computing Laboratory  
Los Alamos National Laboratory  
Los Alamos, NM 87545*

## Abstract

Current network traffic trace generation tools are able to monitor network traffic as it appears on the network, but are incapable of capturing traffic as it progresses through a running protocol stack. Thus, the current generation of tools can not record true application traffic demands, and are incapable of capturing run-time protocol state information to help fine-tune network performance. They also lend no insight into the modulating behavior of some protocols (e.g., TCP), which has been shown to have significant impact on network performance.

In this paper, we introduce the concept of a network *application* traffic trace generation tool. We discuss the need for such a tool, and the necessary modifications to the Linux kernel to enable such a tool. We also describe our own implementation, which we call the Monitor for Application-Generated Network Traffic (MAGNeT), and propose several ways in which MAGNeT can help increase network performance.

## I. Introduction

The networking community routinely uses traffic libraries such as `tcplib` [1], network traces such as those found at the Internet Traffic Archive [2] or the Internet Traffic Data Repository [3], or mathematical models of network behavior such as those discussed in [4] to test the performance of network-protocol enhancements and other network designs.

However, such libraries, traces, and models are based on measurements made either by host-based tools such as `tcpdump` [5] and CoralReef [6] or by global network-mapping tools such as and NLANR's Network Analysis Infrastructure [7]. These tools are only capable of capturing traffic an application sends on the network *after* the traffic has passed through the operating system's protocol stack (e.g., TCP/IP). Tinnakornsriruphap and Feng [8–10] suggest that application traffic experiences significant modulation by the protocol stack before it is placed on the network. This implies that current tools are only able to capture traffic which has already been modulated by a protocol stack; the pre-modulation traffic patterns are unknown.

In order to determine pre-modulation application traffic patterns, as well as determine the modulation experienced by traffic progression through protocols, we offer the Monitor for Application-Generated Network Traffic (MAGNeT). MAGNeT captures traffic (1) generated by applications, (2) passing through each layer (e.g., TCP to IP) of the Linux protocol stack, and (3) entering and leaving the network. Thus, MAGNeT differs from existing tools in that it monitors traffic not only as it enters and leaves the network, but also at the application level and throughout the entire protocol stack. We are aware of two tools which attempt similar measurements.

One tool that appears similar to MAGNeT is the TCP

kernel monitor from Pittsburgh Supercomputing Center [11]. However, MAGNeT differs from the TCP kernel monitor in several ways. First, MAGNeT is able to be used anywhere in the protocol stack, and with any protocol (with very minor alterations to the protocol's code), while PSC's tool is a TCP-specific solution. Additionally, MAGNeT monitors a superset of the data that the TCP kernel monitor does and operates under Linux 2.4.x, whereas the TCP kernel monitor currently works only on NetBSD.

Bolliger and Gross describe a method of extracting network bandwidth information per TCP connection under BSD in [12]. While their research implementation appears to have a similar architecture to MAGNeT, it records only the specific information needed to compute estimated bandwidth for TCP connections because it was written primarily to advance their research in other, related areas. In fact, Bolliger and Gross use results obtained from their tool to argue that network application performance could be improved with the establishment of a tool such as MAGNeT, which goes beyond their simple data extraction.

## II. Software Architecture

MAGNeT consists of both Linux kernel modifications and user application programs. In order to accurately mark events occurring throughout the protocol stack, MAGNeT must exist within the kernel; that is, there must be hooks in the protocol stack code to allow MAGNeT to record events at certain points. Running in the kernel also has the advantage of being able to acquire application traffic patterns with unmodified applications (e.g., there is no need to re-compile or re-link against a special instrumented network library). However, in order to provide the maximum amount of flexibility in data acquisition and processing, the ability to start and end traffic

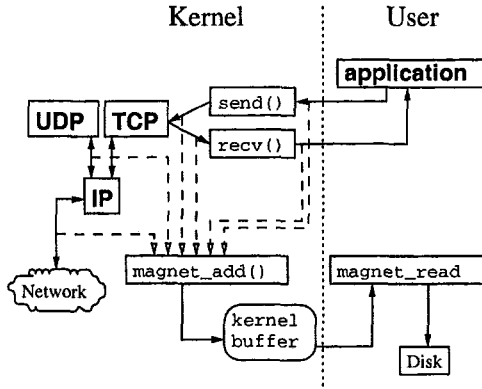


Fig. 1. Overview of MAGNeT Operation

monitoring should be controlled by the user. Therefore, while MAGNeT collects data in the kernel, the data is placed in a special memory region to be read and used by a separate user application program.

The dataflow in a system running MAGNeT is shown in Figure 1. Unmodified applications run as normal on the host system, periodically making use of the network communication routines in the kernel (such as the `send()` and `recv()` system calls). These kernel routines, in turn, make use of TCP, IP, or other network protocols to transfer data on the network. Under MAGNeT, each time a kernel network stack event (`send()`, entering TCP, entering IP, etc.) occurs, the function `magnet_add()` is also called by the kernel. This procedure saves relevant data to a circular buffer in kernel space, which is then saved to disk by an user-level application program (`magnet-read` is provided with the MAGNeT distribution for this purpose, but could easily be replaced by a more sophisticated application).

## A. MAGNeT in Kernel Space

The core functionality of MAGNeT is contained in the new kernel source file `net/magnet/magnet.c`. The function `magnet_add()`, defined in `magnet.c`, adds a data point to a circular buffer that is pinned in physical memory. This function is designed to be very lightweight so that it can be called at several points in the protocol stack without inducing a significant amount of overhead in the protocol processing (see Section III-E for an analysis). Other code in `magnet.c` sets up the circular buffer and the necessary hooks for the user-application program to read data from the buffer. In addition, a new item is added to the file space at `/proc/net/magnet`. This file may be read by any user to determine the current state and parameters of the MAGNeT kernel code.

### A.1 Instrumentation Record

The C header file `include/linux/magnet.h` contains the global definitions for the MAGNeT system.

```
struct magnet_data {
    void *sockid;
    unsigned long long timestamp;
    unsigned int event;
    int size;
    union magnet_ext_data data;
}; /* struct magnet data */
```

Fig. 2. The MAGNeT Instrumentation Record

At the heart of the system is the instrumentation record structure shown in Figure 2.

The instrumentation record is the data structure that `magnet_add()` adds to the kernel buffer at each instrumentation point. `sockid` is a unique identifier for each connection stream<sup>1</sup>, giving MAGNeT the ability of separating data traces into individual streams while protecting user privacy. The `timestamp` field contains a CPU cycle count which serves as both a high-fidelity time measurement for MAGNeT traces, and a synchronization flag between the user and kernel MAGNeT processes (See Section II-B.1). Valid values for the event field (e.g., `MAGNET_IP_SEND`) are given by an enum declaration at the beginning of `magnet.h` and indicate what type of the event is being recorded. `size` is the number of bytes transferred during a specific event.<sup>2</sup> The data field (a optional field selected at kernel compilation time) is a union of various structures in which information specific to particular protocols can be stored. This field provides a mechanism for MAGNeT to record protocol state information along with event transitions.

### A.2 Instrumented Events

MAGNeT is designed to be extensible with regards to the specific events that are monitored. The current distribution instruments the general socket-handling code, the TCP layer, and the IP layer. Other protocols can be easily instrumented by adding new MAGNeT event codes to the enum definition in `magnet.h` and placing calls to `magnet_add()` at appropriate places in the protocol stack. Thus, the mechanisms provided by MAGNeT (that is, generating application-level traces as well as intercepting protocol stack events) are available to all existing and future Linux networking protocols.

Our current MAGNeT distribution records events when the socket handling code receives data from an application, when the TCP layer receives data from the socket handling code, when the IP layer receives data from TCP, and, finally, when IP hands the data off to the network device driver. A similar set of events is recorded for the receive pathway.

Without the optionally-compiled data field, MAG-

```

struct magnet_tcp {
    /* data from "struct tcp_opt" in
       include/net/sock.h */

    unsigned short source;
    /* TCP source port */
    unsigned short dest;
    /* TCP destination port */

    unsigned long snd_wnd;
    /* Expected receiver window */

    unsigned long srtt;
    /* smothed round trip time << 3 */
    unsigned long rto;
    /* retransmit timeout */

    unsigned long packets_out;
    /* Packets which are "in flight" */
    unsigned long retrans_out;
    /* Retransmitted packets out */

    unsigned long snd_ssthresh;
    /* Slow start size threshold */
    unsigned long snd_cwnd;
    /* Sending congestion window */

    unsigned long rcv_wnd;
    /* Current receiver window */
    unsigned long write_seq;
    /* Tail+1 of data in send buffer */
    unsigned long copied_seq;
    /* Head of yet unread data */

    /* TCP flags*/
    unsigned short fin:1, syn:1, rst:1,
                  psh:1, ack:1, urg:1, ece:1, cwr:1;
}; /* struct magnet_tcp */

struct magnet_ip {
    unsigned char version;
    unsigned char tos;
    unsigned short id;
    unsigned short frag_off;
    unsigned char ttl;
    unsigned char protocol;
}; /* struct magnet_ip */

```

Fig. 3. MAGNeT Extended Data for TCP and IP

NeT records only the timestamp and associated data size for each transition between network stack layers. With the data field compiled in, MAGNeT records more extensive data about the instantaneous state of the protocol being monitored. This data typically contains all protocol header information as well as run-time protocol state variables which are not usually available outside of experimental situations. As an example of the kind of information stored within the data field, Figure 3 shows the union members for TCP and IP events.

### A.3 Event Loss

Since the kernel portion of MAGNeT saves events to a fixed-sized buffer, there is a possibility that events may occur when the buffer is full. In this case, MAGNeT is unable to save the event. MAGNeT keeps track of the number of events not recorded, and reports this information as soon as possible (See Section II-B.1).

Our experience to date indicates that unrecorded instrumentation records rarely occur during the monitoring of actual users. MAGNeT provides the capability of tuning its operation, trading between resource utilization and performance. As discussed in [13], with appropriately tuned values MAGNeT is able to record events with less than a 1% loss when a sender saturates a 100Mbps network for a sustained time. Since the majority of users do not approach continuous network-saturation levels, MAGNeT efficiently records virtually all application-generated network traffic.

## B. MAGNeT in User Space

The MAGNeT user interface is designed to be modular; that is, as long as the MAGNeT API is followed, any application can be a MAGNeT user-level application. Thus, this section first discusses the basic elements required of a MAGNeT user-level application and then describes the user applications supplied with the current MAGNeT software distribution.

### B.1 User/Kernel Interface and Synchronization

The MAGNeT additions to the kernel export the circular buffer to user-space applications via kernel/user shared memory. That is, a device file<sup>3</sup> serves as a user-level handle to the kernel shared memory region. Opening this file causes Linux to create a mapping between the kernel memory region and the user address space. With this mapping in place, no additional kernel code is executed; the application program simply reads the shared memory and writes it to disk.

Because the kernel and user processes share the same area of physical memory, they must have a means of synchronization. This is accomplished by using the timestamp field of the instrumentation record as a synchro-

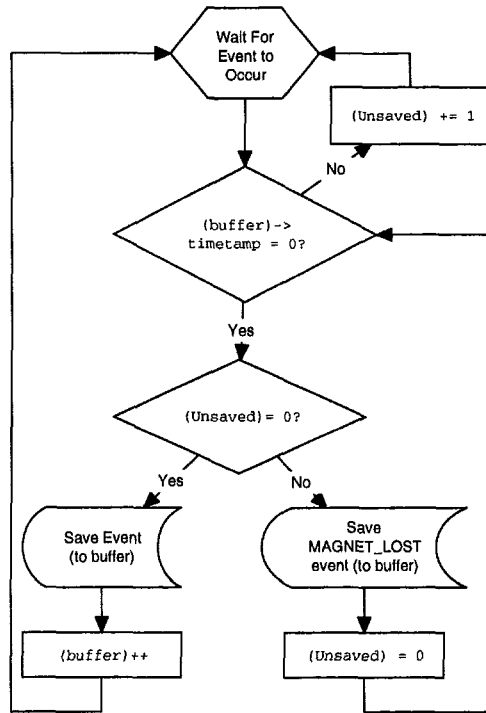


Fig. 4. MAGNeT Kernel Operation

nization flag between the MAGNeT user and kernel processes, as shown in Figures 4 and 5.

Before writing to a slot in the circular buffer, the MAGNeT kernel code checks the value of the timestamp field for that slot. A non-zero value indicates that the slot has not yet been copied to user space and that the kernel buffer is full. In this case, the kernel code increments a count of the number of instrumentation records that could not be saved due to the buffer being full. Otherwise, the kernel code writes a new instrumentation record and advances its pointer to the next slot in the circular buffer.

The user application accesses the same circular buffer via kernel-user shared memory. It also maintains a pointer to its current slot in the buffer. When the timestamp field at this slot becomes non-zero, the application reads the entire record, saves it to disk, and sets the timestamp field back to zero to signal the kernel that the slot is once again available. It then advances its pointer to the next slot in the circular buffer.

If the kernel has a non-zero count of unsaved events and buffer space becomes available (i.e., the timestamp field of the kernel's currently active slot is set to zero by the user application), the kernel writes a special instrumentation record with an event type of MAGNET\_LOST and with the size field set to the number of instrumentation records that were not recorded. Thus, during post-processing of the data, the fact that events were lost is detected at the appropriate chronological

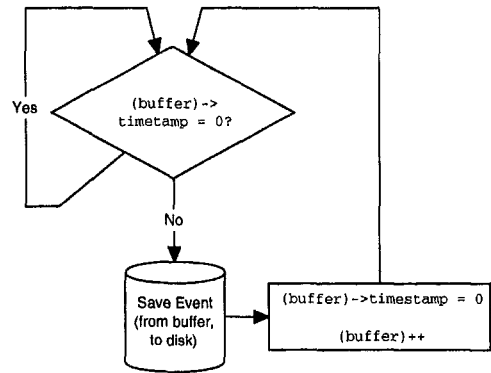


Fig. 5. MAGNeT User Operation

place in the data stream.

## B.2 MAGNeT System Information

In order to accurately gauge the amount of time spent in protocol-stack layers, MAGNeT requires very high-fidelity timing. To this end, MAGNeT timestamp values are recorded in units of processor clock cycles by the kernel's `get_cycles()` function. In addition, the first record stored by MAGNeT in the circular buffer is a record of type `MAGNET_SYSINFO`, whose `size` field contains the processor clock speed estimated by the kernel at boot time. This information allows for easy conversion of MAGNeT timestamp values to wall-clock time.

The `MAGNET_SYSINFO` event record also allows MAGNeT to be endian-aware. Since this field is guaranteed to be the first record in the circular buffer every time the MAGNeT device file is opened, it will be the first record that the user-application reads. The value of the event field of this record lets data processing software determine if the records were saved on a big- or little-endian machine (e.g., if the records are saved to a file for later processing on a different machine). Specifically, `MAGNET_SYSINFO` is defined in `magnet.h` to have a value of `0x01234567`. If the first record read by the data processor has an event field of this value, no endian translation is necessary. On the other hand, if the first record's event field contains a different value (e.g., a value of `0x67452301`), the file was saved on a machine with a different endian orientation then the processing machine, so endian translation is necessary.

## B.3 `/proc/net/magnet`

The `/proc/net/magnet` file allows user applications to determine the state of the MAGNeT kernel process. The existence of this file serves as proof that the MAGNeT code is active in the kernel. This file contains information such as the major and minor numbers for the MAGNeT shared memory device file, the size of the circular buffer, and other information that may be useful to

user-level applications.

## B.4 Current MAGNeT User Implementation

As provided in our current distribution of MAGNeT, the user interface consists of three programs, `magnet-read`, `mkmagnet`, and `magnet-parse`, along with a couple of scripts to ease automated traffic trace generation and collection.

`magnet-read` is the primary means of obtaining MAGNeT traffic traces; its function is to read the data from the kernel's circular buffer. Our first version of `magnet-read` copied records out of the shared memory and wrote them to a file on disk. We found, however, that this approach was unable to keep up with the demands of a simple test application that tries to saturate a 100-Mbps Ethernet network. Instead, the current version of `magnet-read` uses the memory-mapped I/O features in the Linux kernel. Once an empty "binary trace file" exists, `magnet-read` maps this file into its memory space, and then saves data to disk by simply performing a memory copy between the kernel-user shared memory and the memory region mapped to the binary trace file. This approach reduces overhead significantly and allows MAGNeT to record data on even high-speed networks with little chance of record loss. The `mkmagnet` application creates and initializes the binary trace file prior to it being mapped into memory by `magnet-read`. Finally, `magnet-parse`, reads data collected by `magnet-read` and dumps a tab-delimited ASCII table of the collected data for further processing, performing endian translation as necessary.

The MAGNeT distribution also includes two shell scripts which allow network administrators to create an automated application-monitoring environment. `magnet.cron`, the overall MAGNeT management script, ensures that the MAGNeT device file exists and that a binary trace file has been created by `mkmagnet`. Additionally, if invoked while `magnet-read` is running, `magnet.cron` terminates the current MAGNeT data collection session and calls `magnet.copy` to transfer the data to a remote archive.<sup>4</sup> Before exiting, `magnet.cron` starts `magnet-read` as a background process to save network events to disk. Thus, the management script may be added as a crontab event (e.g., run everyday at midnight) to collect data on a diverse set of machines without requiring special action by the users of the network.

## III. MAGNeT Performance

In this section, we determine the effect of running MAGNeT through a variety of tests. We compare overall attainable bandwidth on a system running MAGNeT to that of a system running `tcpdump` as well as a sys-

tem running no monitoring software. We also look at CPU utilization under these conditions, and at the effect of MAGNeT on real-time traffic streams such as multimedia traffic. We conclude the section with a brief discussion of how the different design decisions made in MAGNeT and other monitors result in the observed performance differences.

## A. Experimental Environment

We use a common environment for all the tests discussed in this section. This environment consists of two identical dual 400-MHz Pentium IIs connected to each other via an Extreme Networks Summit 7i Gigabit Ethernet switch. Each machine contains 128MB of RAM, ATA-33 IDE hard drives and both 100-Mbps (Net-Gear) and 1000Mbps (Alteon) Ethernet cards. All non-essential Linux services are disabled on the test machines, and no extraneous traffic is allowed on the network. MAGNeT is set to record a minimal set of information per event (i.e., the `data` field is not compiled into the MAGNeT build).

## B. Network Throughput

As an indication of how much MAGNeT affects network applications, we measure the maximum data rate between a sender and receiver. We also measure the overhead of running `tcpdump` as a point of comparison.

In total, we run six different configurations, each on 100-Mbps and Gigabit Ethernet networks. The first configuration, our baseline, runs between two machines with stock Linux 2.4.3 kernels. The second configuration uses the same machines but with the MAGNeT patches installed on both sender and receiver. Although present in the kernel, MAGNeT instrumentation records are not saved to disk. The third configuration is the same as the second except `magnet-read` runs on the receiver to drain the kernel-event buffer. The fourth configuration is also the same as the second except `magnet-read` runs on the sender. For the fifth and sixth configurations, we run `tcpdump` on either the sender or the receiver, with a stock Linux 2.4.3 kernel (i.e., no MAGNeT code installed). For each trial we run `netperf` [14] on the sender to transmit data as fast as possible.<sup>5</sup>

Table I shows the results of our bandwidth experiments. Along with the mean, the width of the 95% confidence interval is given. As shown in this table, MAGNeT never reduces the achievable network bandwidth by more than 4.5%. By comparison, while `tcpdump` has roughly the same impact on performance for current networks, it suffers dramatically as network speeds increase. Thus, we conclude that MAGNeT is better able to adapt to tomorrow's networking infrastructure than the current version of `tcpdump`.

It is worth noting that these comparisons are not en-

TABLE I  
NETWORK THROUGHPUT REDUCTION

Configuration	Fast Ethernet (100Mbps)		Gigabit Ethernet (1000Mbps)	
	Throughput	% Reduction	Throughput	% Reduction
Linux 2.4.3	94.1 $\pm$ 0.0	-	459.5 $\pm$ 1.6	-
MAGNeTized	94.1 $\pm$ 0.1	0.01	452.5 $\pm$ 1.8	1.53
magnet-read/rcv	90.8 $\pm$ 0.8	3.56	444.3 $\pm$ 1.7	3.30
magnet-read/snd	90.7 $\pm$ 0.9	3.67	440.2 $\pm$ 2.1	4.19
tcpdump/rcv	89.4 $\pm$ 1.5	5.04	290.7 $\pm$ 15.6	36.74
tcpdump/snd	89.4 $\pm$ 0.8	5.42	343.2 $\pm$ 18.7	25.30

tirely fair. As discussed in Section III-F, MAGNeT and tcpdump are designed to record different (but similar) sets of information. However, since no tool exists which captures the same information as MAGNeT, we use tcpdump as the closest commonly-available tool.

By default (and as used in our experiments), tcpdump stores the first 68 bytes of every packet. During these tests, the MAGNeT per-event record size is 24 bytes. However, since MAGNeT instruments the entire network stack, it records the packet's transitions between Application, TCP, and IP, as well as transmission on the network. Thus, although MAGNeT stores approximately 1/3 less data than tcpdump *per event*, it records at least 3 events per packet, as compared to tcpdump's one record per packet. Hence, the total data saved by MAGNeT per packet is no less than 72 bytes, and may be significantly more (depending on the fragmentation to the data in the packet caused by the protocol).

### C. CPU Utilization

Under Linux, netperf estimates CPU load by creating a low-priority process which increments a counter. This process, being the lowest priority task in the system, should only execute when the CPU has nothing else to execute, so the counter is only incremented when the CPU would otherwise be idle. Thus, a low counter value implies a high CPU utilization, and a high counter value implies low CPU utilization. Using this feature with the above set of tests, we estimate the additional CPU load incurred by both MAGNeT and tcpdump. The increase in CPU load, averaged over the sender and receiver, during the above tests is shown in Figure 6.

As can be seen, MAGNeT uses proportionally less CPU than tcpdump, which is expected given the results of our bandwidth tests. Also apparent is a decrease in CPU utilization when moving from 100-Mbps to 1000-Mbps. This drop is a result of the fact that our Gigabit Ethernet cards perform interrupt coalescing by default. That is, they wait for several packets to arrive from the

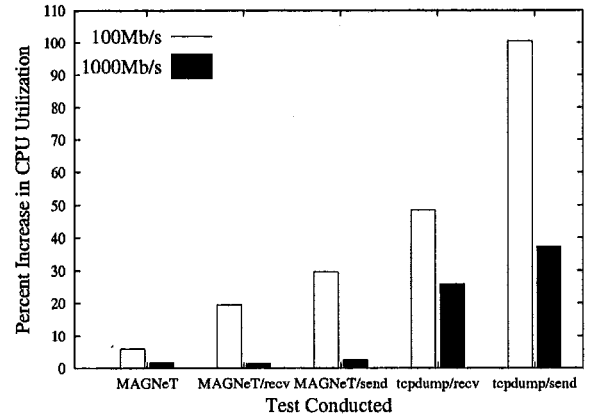


Fig. 6. Average Percent Increase in CPU Utilization

network before interrupting the CPU. Thus, the cost of servicing the network device interrupt is amortized over several packets. This reduces the total amount of work performed by the CPU, as shown in Figure 6. Had interrupt coalescing been disabled, the average CPU utilization for both MAGNeT and tcpdump would have increased.

### D. Streaming MAGNeT

In order to determine what visible effect the operation of MAGNeT has on streaming media we set up a web server on one of our test machines to stream an 8-minute, 51-second MPEG clip of *Crocodile Dundee*. We then viewed the clip with MAGNeT running only on the server, with MAGNeT executing only on the client, and with MAGNeT not executing at all.

Our results are summarized in Figure 7. Note that most streaming MPEG clients buffer data to maintain a constant framerate. This fact causes frames-per-second to be an imprecise measure of streaming network performance. Therefore, the metric we have chosen is the total wall-clock time taken for the entire clip to be sent to the



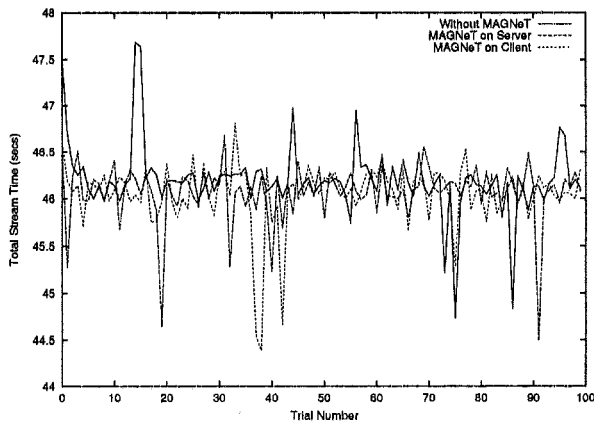


Fig. 7. Streaming MAGNeT Effects

client.

As can be seen, MAGNeT has minimal effect on MPEG streaming. Over 100 trials, the average time to stream the movie clip without MAGNeT is 46.02 seconds, with a 95% confidence interval of  $\pm 0.07$  seconds. To transfer the movie clip with MAGNeT running on the server took an average 46.07 seconds, and with MAGNeT on the client, 46.05 seconds. (Both of the MAGNeT cases have a 95% confidence interval of  $\pm 0.06$  seconds.)

## E. Network Perturbation

Adding CPU cycle-counting code in `magnet_add()` allows us to determine the amount of time taken to save a single event to the buffer. In a similar manner (by instrumenting the relevant areas of `magnet-read`), we can determine the average time taken to save an event from the buffer to disk. The sum of these values is the amount of time taken by MAGNeT to record events.

Our tests indicate `magnet_add()` uses 556 cycles per recorded event while `magnet-read` uses 425 cycles. So, on our 400-MHz machines, MAGNeT takes  $(556 \text{ cycles} + 425 \text{ cycles}) / (400 \text{ Mcycles/second}) = 2.4 \mu\text{sec}$  to record each event. However, on a 100-Mbps Ethernet, a minimal TCP packet (that is, a packet of 40-bytes) will take at least  $(40 \text{ bytes} \times 8 \text{ bits/byte}) / 100 \text{ Megabits/second} = 3.2 \mu\text{sec}$  to transfer. This comparison suggests MAGNeT-induced disturbances into TCP traffic streams should be quite small.<sup>6</sup>

## F. Design of `tcpdump` vs. MAGNeT

While `tcpdump` and MAGNeT are similar in that they are both monitors of network traffic, they are also very different in that they monitor different aspects of the traffic and are based on different design philosophies.

`tcpdump`, like `magnet-read`, is a user-interface application relying on functionality contained in a lower layer. In this case, the lower layer is called `libpcap`,

which has been successfully used with other applications, such as CoralReef. The critical difference is that while MAGNeT operates largely within the Linux kernel, `libpcap` is implemented as a library working in user space under a variety of operating systems.

The exact method used by `libpcap` to intercept network packets varies depending on the features available in the root operating system, but always involves a system call or other facility to cause a switch into kernel mode and a copy of memory from the kernel to the user-level library. This call-and-copy is repeated for every packet traveling across the interface being monitored. At high network speeds (and thus high packet-transfer rates), the overhead of copying each individual packet between kernel and user space becomes a significant burden. MAGNeT benefits from having code embedded in the kernel to aggregate multiple network packets into a single space which then is copied in bulk, thus amortizing the cost of the copy over multiple packets. This approach incurs less overhead but is not as portable as `libpcap`'s method.

Finally, we note that the kind of data collected by `tcpdump` and MAGNeT are not exactly the same. As used in the experiments in this paper,<sup>7</sup> MAGNeT collects just packet generation time and data size. `tcpdump`, on the other hand, collects packet time information along with a sampling of the actual data contained in the packet. MAGNeT ignores this data mostly out of privacy concerns.

## IV. Applications of MAGNeT

MAGNeT provides network practitioners with the ability to discover true application traffic demands while maintaining application transparency. This ability has many practical applications, some of which we will discuss in this section.

### A. Network Security

A standard method of detecting network intrusion is to have an automated system continually watching network traffic patterns and flagging anomalous behavior for a human operator to investigate. This approach requires all traffic on the network to flow through a centralized monitoring station, which not only introduces a single point of failure to the network but also provides a potential bottleneck that may reduce achieved network bandwidth significantly (while, at the same time, increasing network latency).

MAGNeT provides an alternative solution. We have shown that MAGNeT, unlike `tcpdump`, runs almost transparently for most applications, even on high-speed networks. Thus, MAGNeT may be deployed on every computer in an installation. If this is the case, there is no need for all traffic to flow through a central monitoring

machine. Instead, each machine may collect its own traffic patterns and then periodically have `magnet.cron` send its collected data to a central processor. This processor is then able to analyze campus-wide network activity, with a finer granularity than currently available.<sup>8</sup> Unlike current solutions, if for some reason the central processor goes down, the rest of the computers on the network are able to operate without difficulty. Thus, the problems of a single network traffic sink are eliminated.

## B. Protocol Tuning and Troubleshooting

With the optional data file compiled in, MAGNeT has the ability to return snapshots of complete protocol state (information previously only available under simulation environments) during execution of real applications on real-world networks. This kind of data is a powerful tool for aiding the debugging and fine-tuning of network protocol implementations.

## C. Traffic Pattern Analysis

One application of MAGNeT is to investigate differences between the traffic generated by an application and that same traffic but after modulation by the protocol stack, i.e., when the traffic hits the network. An example of this kind of modulation is shown in Figure 8. This figure is the MAGNeT trace of using FTP to send a Linux 2.2.18 bzipped tar file from our facilities in Los Alamos, NM to a location in Dallas, TX. As can be seen by examining the graph, the FTP application attempts to send 10KB segments of data every 1/5 of a second. However, the Linux protocol stack (TCP and IP in the case of FTP) modulates this traffic pattern into approximately 1500 byte packets at considerably shorter intervals.

It may be assumed that since the maximum data size on an ethernet network is 1500 bytes, the protocol stack is simply modulating the data to this size to obtain valid ethernet traffic. However, if we send the traffic stream *as it was delivered to the network* through another TCP stack (this is exactly what network researchers do when they use a `tcpdump`-derived traffic trace as input to a network simulation), we again see modulation. Every successive run of network-delivered traffic through TCP further modulates the traffic, as shown in Table II. The table reflects the average data size of output TCP packets and the average time in seconds between output TCP packets, using the data stream from the previous TCP output as input. Likewise, Figure 9 shows the effect of successive TCP stacks on achieved bandwidth across the same WAN pathway used for the original FTP transfer.

One real-life instance where traffic is subjected to multiple TCP stacks is in the case of firewalls which break a TCP connection into two connections — one “inside” connection between the internal host and the firewall and one “outside” connection between the firewall and the

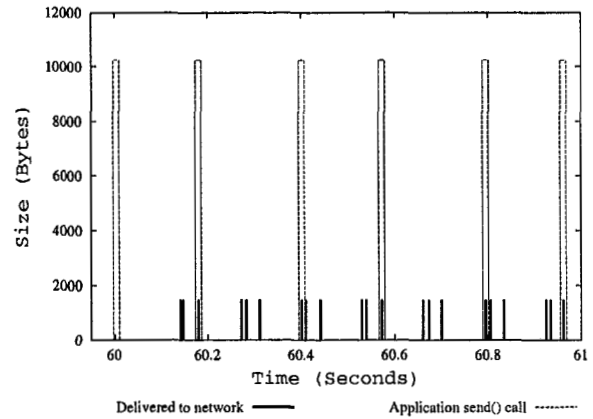


Fig. 8. MAGNeT FTP trace

TABLE II  
EFFECT OF MULTIPLE TCP STACKS

Trial	Data Size	Interpacket Spacing (sec)
Application	3284	0.124
1st TCP stack	1016	0.045
2nd TCP stack	919	0.037
3rd TCP stack	761	0.079
4th TCP stack	723	0.122

external world. Imagine two locations, AS1 and AS2, both have this kind of active firewall installed. This requires a FTP connection from a host within AS1 to a host within AS2 to actually consist of three connections: (1) from the AS1 internal host to the AS1 firewall, (2) from the AS1 firewall to the AS2 firewall (across the internet), and (3) from the AS2 firewall to the AS2 internal host. Thus, the traffic pattern generated by the FTP application passes through three different TCP stacks.

As can be seen in Table II, successive TCPs reduce the data size of each packet in an exponential fashion. In addition, the first TCP stack (that is, the TCP stack used by the application) radically reduces the inter-packet spacing, while each successive TCP stack (e.g., the TCP stacks of a network simulation or active firewall) slowly increases the inter-packet delay. All of this has the effect of drastically reducing the achieved bandwidth during actual transfers, as reflected in Figure 9. After just three TCP stacks, the achievable bandwidth has been reduced by 76%.

## D. Application-Generated Trace Library

Currently existing models of network traffic have been developed using network traffic traces gathered via traditional network monitoring systems. These models are then used to develop new network protocols and net-

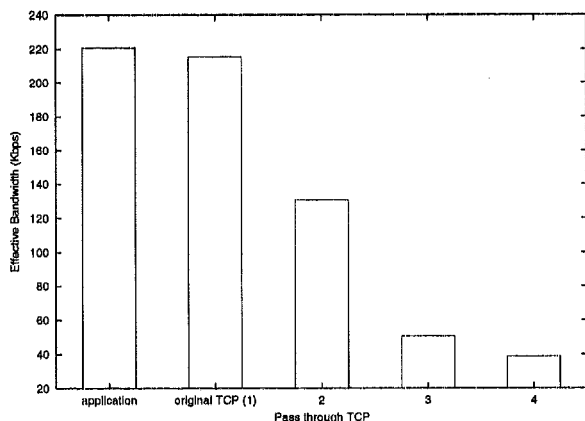


Fig. 9. Effect of multiple TCP stacks

working enhancements.

However, the kind of results typified by Figure 8 indicate network traffic demands of applications are not accurately reflected by traffic on the network wire, since the traffic pattern has already been modified by the current network protocol. Hence, while current network models may accurately reflect current network-wire traffic, they are not useful in optimizing application communications.

We have seen (in Section IV-C) that recursively passing traffic through TCP stacks results in decreased network performance. Yet the current tools available to network researchers (such as `tcpdump`) only allow traffic collection after that traffic has passed through at least one TCP stack. Thus, the results of simulations using this traffic data are corrupt; the input data has already been modulated. This is a critical element for developing next-generation high-speed network protocols.

Traffic traces generated by MAGNeT provide a realistic picture of the protocol-independent traffic demands generated by applications running on today's networks. Thus, MAGNeT provides network researchers and developers with a better understanding of the requirements of future networks.

## V. Future Work

MAGNeT currently exists as a prototype implementation, and as such, its user interface is not highly refined. We would like to improve the interface by allowing the user to set various MAGNeT parameters (i.e., the kinds of events to be recorded, the size of the kernel buffer, etc.) at run-time rather than at kernel compile-time. This could be accomplished by making the current `/proc` file writable and would greatly increase the usability and flexibility of MAGNeT.

Another potential area of improvement in MAGNeT is the mechanism used to store saved data from the kernel buffer to disk. As it is currently implemented, the mechanism works but requires a user well-versed in how

to operate MAGNeT (or a script which takes care of the details for the user). A better approach may be to utilize kernel threads to perform all steps of the instrumentation. With this methodology, the need for the special device file, the file created by `mkmagnet`, and the kernel/user shared memory would be eliminated. In addition, kernel threads may lower MAGNeT's current low event loss rate by reducing the need for a context switch to save data. However, the execution of kernel threads could break the application-usage-pattern transparency which MAGNeT currently is able to achieve. Additionally, kernel threads may remove the ability of easily integrating MAGNeT data collection facilities into new user applications. The use of kernel threads may be explored for future versions of MAGNeT, along with other options for improving this interface, such as the Turbo Packet scheme used by Alexey Kuznetsov to increase `tcpdump`'s performance under Linux.

Lastly, performing timing with the CPU cycle counter can be problematic on contemporary CPUs which are able to change their clock rate in response to power management policies. If the kernel could detect such a change, MAGNeT could easily hook into the clock-rate change detection code and output a new `MAGNET.SYSINFO` event with the new timing information. This would keep timing relatively consistent across CPU clock-rate changes. However, there is currently no way for production Linux kernels to detect CPU clock rate changes at run-time.<sup>9</sup>

## VI. Conclusion

We have developed a new kind of monitor for the Linux networking community. Our Monitor for Application-Generated Network Traffic (MAGNeT) collects run-time data about the network protocol stack as well as application traffic demands before modulation by any protocol stack. In addition, MAGNeT is able to collect this data while maintaining user- and application transparency. Thus, MAGNeT may be used on live systems to obtain real-world application traffic traces and protocol state information in production environments.

This data may be used, among other things, for debugging existing protocol implementations, understanding possible performance degradation seen under various network architectures, designing new networking protocols specifically to take advantage of true application traffic patterns, and developing more realistic models of network traffic. The data collection capabilities of MAGNeT also have potential use in fields such as network security which rely on a comprehensive understanding of the traffic existing in an institutional network.

The combination of abilities offered by MAGNeT make it a valuable tool to network designers, implementors, researchers, and administrators. We intend for

MAGNeT development to continue and lead to further advances in high-performance networking.

## Availability

A distribution of MAGNeT, consisting of the Linux 2.4 kernel patch, the simple user-application program `magnet-read`, and supporting material, is available from <http://www.lanl.gov/radiant> under the GNU Public License (GPL). Also available on this site are other published and unpublished documents relating to MAGNeT.

## Notes

<sup>1</sup>The `sockid` field is, in fact, the run-time value of the pointer to the kernel's status information for the specific connection.

<sup>2</sup>A negative value in the size field reflects the error code returned by the function causing the event.

<sup>3</sup>The major and minor numbers for this device are system specific and can be discovered by inspecting `/proc/net/magnet`.

<sup>4</sup>Since `magnet.copy` is called while `magnet-read` is not running, any traffic produced by the data archiving will not be captured by MAGNeT. This behavior can easily be changed by re-ordering the commands in `magnet.cron`.

<sup>5</sup>The command used was "`netperf -P 0 -c {local CPU index} -C {remote CPU index} -H {hostname}`"

<sup>6</sup>Of course, with 1000-Mbps networks, the time taken to send a minimal TCP packet is reduced by an order of magnitude. In this case, MAGNeT takes significantly longer to record an event than the transmission time. However, it is likely that Gigabit networks will be used with much faster machines than our test machines, thus reducing this discrepancy.

<sup>7</sup>By utilizing the data field of the instrumentation record, MAGNeT is able to capture much more detailed traces, but still not actual packet data.

<sup>8</sup>Since MAGNeT never collects actual data (only traffic patterns), the privacy of each individual machine is maintained.

<sup>9</sup>Dynamic CPU clock changing is also an issue with many aspects of the Linux kernel. For instance, short delay loops in the kernel rely on  $x$  number of loops taking  $y$  amount of time to compute. This relation is also affected by CPU clock-rate changes.

## References

- [1] P. Danzig and S. Jamin, "tcplib: A Library of TCP Internetwork Traffic Characteristics," <http://irl.eecs.umich.edu/jamin/papers/tcplib/tcplibtr.ps.z>, 1991.
- [2] "The Internet Traffic Archive," <http://ita.ee.lbl.gov/html/traces.html>.
- [3] A. Kato, J. Murai, and S. Katsuno, "An Internet Traffic Data Repository: The Architecture and the Design Policy," in *INET'99 Proceedings*.
- [4] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, June 1995.
- [5] "tcpdump," <http://www.tcpdump.org>.
- [6] CAIDA, "CoralReef Software Suite," <http://www.caida.org/tools/measurement/coralreef>.
- [7] A.J. McGregor, H-W Braun, and J.A. Brown, "The NLANR Network Analysis Infrastructure," *IEEE Communications*, May 2000.
- [8] P. Tinnakornsrisuphap, W. Feng, and I. Philp, "On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System," in *Proc. of the Int'l Conf. on Dist. Comp. Sys.*, April 2000.
- [9] W. Feng and P. Tinnakornsrisuphap, "The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems," in *Proc. of the Int'l Conf. on Parallel Processing*, August 2000.

- [10] W. Feng and P. Tinnakornsrisuphap, "The Failure of TCP in High-Performance Computational Grids," in *Proc. of SC 2000: High-Performance Networking and Computing Conf.*, November 2000.
- [11] J. Semke, "PSC TCP Kernel Monitor," Tech. Rep. CMU-PSC-TR-2000-0001, PSC/CMU, May 2000.
- [12] J. Bolliger and R. Gross, "Bandwidth Monitoring for Network-Aware Applications," *Proc. of the 10th Annual Int'l Symposium on High Performance Distributed Computing*, August 2001.
- [13] W. Feng, J. R. Hay, and M. K. Gardner, "MAGNeT: Monitor for Application-Generated Network Traffic," *To appear in Proc. of the 10th Annual Int'l Computer Communication and Networking Conf.*, October 2001.
- [14] "Netperf," <http://www.netperf.org>.